**Small XPath Quiz**

Can you give an expression that returns the last/first occurrence of each distinct price element?

```
<price>3</price>
<price>1</price>
<price>3</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1</price>
<price>7</price>
```

Should return

```
<price>3</price>
<price>4</price>
<price>1</price>
<price>7</price>
```

Should return

```
<price>3</price>
<price>1</price>
<price>4</price>
<price>7</price>
```

*What’s the result for this query: \( \text{descendant::price} = \text{preceding::price}[2] \)*

**Small XPath Quiz**

Can you give an expression that returns the smallest (last) price element?

```
<price>3</price>
<price>1</price>
<price>3</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1</price>
<price>7</price>
```

Should return

```
<price>1</price>
```

Should return

```
<price>3.0</price>
```

What if we mean number-distinctness (not strings)?

**Small XPath Quiz**

Can you give an expression that returns the smallest (last) price element?

```
<price>3.0</price>
<price>1</price>
<price>3.00</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1.000</price>
<price>7</price>
```

Should return

```
<price>1.000</price>
```

Should return

```
<price>3.0</price>
```

**Small XPath Quiz**

Can you give an expression that returns the smallest (last) price element?

```
<price>3.0</price>
<price>1</price>
<price>3.00</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1.000</price>
<price>7</price>
```

Should return

```
<price>1.000</price>
```

Should return

```
<price>3.0</price>
```
Recall: Koch’s CVT-algorithm for full XPath 1.0

**Question** Which subsets of nodes need to appear in the CVTs?

**Answer** (1) Compute top-down for filter-less query-part the exact node set, until a filter appears. (2) Build CVT (bu) for the respective sets.

**Example** /*[sum(preceding::*/@a)>sum(following::*/@a)]

(1) simply use Core-XPath algorithm here to compute in linear time the correct node set.

(2) Now build Context-Value-Tables for this node set only, going bottom-up through the filter.

**Context-Value Tables**

<table>
<thead>
<tr>
<th>Query</th>
<th>Position</th>
<th>Value</th>
<th>Filter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/*</td>
<td>1</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>b.a</td>
<td>1 False</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>5,6,a</td>
<td>2 False</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>6,7,a</td>
<td>1 False</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>7,8,a</td>
<td>2 False</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>8,9,a</td>
<td>3 False</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>9,10,a</td>
<td>3 False</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>10,11,a</td>
<td>3 False</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>11,12,a</td>
<td>3 False</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>11</td>
<td>12,13,a</td>
<td>3 False</td>
</tr>
</tbody>
</table>

Recall: Top-Down Evaluation of Simple Paths

- evaluate in one single pre-order traversal (using a stack)
- Streaming Algorithm: No need to store the document! Can evaluate on SAX event stream.
- True Streaming, with memory need proportional to height.

**Simple Path** //a_1/a_2/a_3/.../a_n
- TIME: one pass through document tree. height is bounded by depth of document tree.
- SPACE: output buffers, if subtrees of match nodes should be printed!

If we print node-IDs, then no output buffers are needed!
Recall: Top-Down Evaluation of Simple Paths

- evaluate in one single pre-order traversal (using a stack)

If we print node-IDs, then no output buffers are needed!

Simple Path: //a_1/a_2/a_3/.../a_n

**TIME**
one pass through document tree

**SPACE**
stack of query positions

height is bounded by depth of document tree.

Streaming Algorithm:

- No need to store the document!
- Can evaluate on SAX event stream.

1 Byte/pos is enough for small queries!

Any good implementation of this algorithm should work for documents with depth up to a couple of millions, and NO restriction on document size!

Arbitrary Slash+Slashslash

- evaluate in one single pre-order traversal (using a stack)

For arbitrary queries with //a/b//c

multiple //’s

Result node!
Mark it, and stay in p=3.

Arbitrary Slash+Slashslash

- evaluate in one single pre-order traversal (using a stack)

For arbitrary queries with //a/b//c

multiple //’s

Stay at position 3, for the complete subtree!
Never go back to pos. 1 or pos. 2!
Arbitrary Slash+Slashslash

→ evaluate in one single pre-order traversal (using a stack)

Arbitrary queries with /, /, /, *
multiple /’s

query match position: p = 3

Optimizations (for Output Buffers)
1. If inside a matched subtree, record position (or range within buffer), instead of creating a new output buffer.
2. If subtree is finished (we are not inside a match), then we can write its buffer out and can start with empty buffer again. (Worst Case: root node selected, size of doc. Needed.)

→ Same as before
Jump back within /-sequence. AT MOST to the beginning of the last /.
Use KMP within /-sequence.
For *’s: build several KMP-tables.

Recall
Deterministic Automaton runs in
→ linear time and
→ constant space
(plus stack of states, if we run on paths of a tree)

1. Automaton Approach

Recall
Deterministic Automaton runs in
→ linear time and
→ constant space
(plus stack of states, if we run on paths of a tree)
1. Automaton Approach

Problem

If it is NOT e here, then what to do??

E.g., a b c d c d

We should be in state X!

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters

Æ for x ≠ c, not important what x is
Æ only x=c / x ≠ c matters
Advantage of automata: can be combined to evaluate MANY queries "in parallel".

Questions
1. Which transition is WRONG?
2. How many transitions are missing?
Advantage of automata:
→ can be combined to evaluate MANY queries “in parallel”.

Q1 = //a/b//c/d/*/e//f/g//h
Q2 = //a/c

Questions
1. Which transition is WRONG?
2. How many transitions are missing?

To evaluate MANY queries “in parallel”.

Question
What is SIZE(A1) wrt size of Q1?

Take
(1) SIZE(A) = # of states
(2) SIZE(A) = # of transitions

ONE look-up per node!
3. The Size of the DFA

Size of DFA = exponential in \*'s
(not a real concern)

Theorem [GMOS'02] The number of states in the DFA for one linear XPath expression P is at most:

$$k + |P| k s^m$$

k = number of //
s = size of the alphabet (number of tags)
m = max number of \* between two consecutive //

How to deal with filters?

When we meet the c-nodes (in pre order traversal) we do not know yet if the filter will evaluate to true.

Must be stored in memory

Optimizations

1. Store potential match trees as DAGs
2. Release potential match trees as early as possible!

Release potential match trees as early as possible!

Find earliest point at which we know the filter is true.
How to deal with filters?

//a[./d/e]/b//c

- Size of largest documents that can be streamed in this way depends on:
  - #filters,
  - sizes of (pre) selected trees,
  - quality of (1), (2), etc..

- Release potential match trees as early as possible!

Find earliest point at which we know the filter is true.

No need to store. Stream!
5. Streaming XPath Algorithms

Some following slides are by T. Amagasa and M Onizuka (Japan)
See http://www.dasfaa07.ait.ac.th/DASFAA2007_tutorial3_1.pdf

Most of the following slides are by Dan Suciu (the above slides are
Actually also based on Suciu’s slides ☺ )
Basic NFA Evaluation

Properties:

- Space = linear
- Throughput = decreases linearly

Systems:

- XFilter [Altinel & Franklin'99], YFilter.
- XTrie [Chan et al.'02]
Basic DFA Evaluation

Properties:
- Throughput = constant!
- Space = GOOD QUESTION

System:
- XML Toolkit [University of Washington]
  http://xmltk.sourceforge.net

The Size of the DFA

Theorem [GMOS’02] The number of states in the DFA for one linear XPath expression $P$ is at most:

$$k + |P| k s^m$$

- $k$ = number of $//$
- $s$ = size of the alphabet (number of tags)
- $m$ = max number of * between two consecutive $//$

Size of DFA: Multiple Expressions

//section//footnote
//table//footnote
//figure//footnote
. . . . .
//abstract//footnote

100 expressions

$2^{100}$ states!!

There is a theorem here too, but it’s not useful…

Solution:
Compute the DFA Lazily

- Also used in text searching
- But will it work for $10^6$ XPath expressions?
- YES!
- For XPath it is provably effective, for two reasons:
  - XML data is not very deep
  - The nesting structure in XML data tends to be predictable

Lazy DFA
DFA, view class: $XP\{/\,/,\,*,\}\

Features
- Sharing the process of / and //, * and tag
- DFA-based
- Compute DFA lazily (on demand)
- # of DFA states
- Independent from # of XPath exprs.
- Depends on DataGuide size (schema)

Issue
- Predicates: XPush machine [SIGMOD’03]
Lazy DFA and “Simple” DTDs

• Document Type Definition (DTD)
  – Part of the XML standard
  – Will be replaced by XML Schema
• Example DTD:
  ```
  <!ELEMENT document (section*)>
  <!ELEMENT section ((section|abstract|table|figure)*)>
  <!ELEMENT figure (table?,footnote*)>
  .  .  .  .
  ```

**Definition** A DTD is simple if all cycles are loops

Lazy DFA and “Simple” DTDs

**Theorem [GMOS’02]** If the XML data has a “simple” DTD, then lazy DFA has at most:

\[ 1 + D(1+n)^d \]

states.

\( n = \text{max depths of XPath expressions} \)
\( D = \text{size of the “unfolded” DTD} \)
\( d = \text{max depths of self-loops in the DTD} \)

**Fact of life:** “Data-like” XML has simple DTDs

Lazy DFA and Data Guides

**Definition [Goldman&Widom’97]** The data guide for an XML data instance is the Trie of all its root-to-leaf paths

**Fact of life:** real XML data has “small” data guide [Liefke&S.’00]

Lazy DFA and Data Guides

**Theorem** “Non-simple” DTDs are useless for the lazy DFA

• “Everything may contain everything”
Lazy DFA and “Simple” DTDs

**Theorem [GMOS’02]** If the XML data has a data guide with $G$ nodes, then the number of states in the lazy DFA is at most:

$$1 + G$$

$G = \text{number of nodes in the data guide}$

---

**Number of Lazy DFA States - SYNTHETIC Data**

- $10^3$ XPath: 4000 states
- $10^4$ XPath: 4000 states
- $10^5$ XPath: 40000 states

**Number of Lazy DFA States - REAL Data**

- $10^3$ XPath: 95 states
- $10^4$ XPath: 95 states
- $10^5$ XPath: 95 states

**Throughput for $10^3$, $10^4$, $10^5$, $10^6$ XPath expressions**

- [prob(*)=10%, prob(//)=10%]

  - Parser: 10MB/s
  - Lazy DFA: 5.4MB/s

---

END Lecture 9